White Paper: Integrating Content Into the Virtual Content Repository

prev next contents view as PDF get Adobe Reader

Integrating Content into the BEA Virtual Content Repository

Overview and Benefits

Portals provide an extremely efficient mechanism for aggregating, delivering and presenting enterprise applications and content. However, without an efficient aggregation mechanism the cost and complexity can be onerous when connecting an enterprise portal to multiple, heterogeneous repositories.

Many large companies have this problem; according to Forrester 78% of companies surveyed have more than one content repository and 43% have six or more. The multitude of repositories in any given enterprise may be comprised of Enterprise Content Management systems, business applications or home-grown solutions.

WebLogic Portal provides a Virtual Content Repository (VCR) that enables customers to plug in multiple, heterogeneous content repositories. Leading ECM vendors such as FileNet, Documentum andFatWire and have built an integration, or Content Service Provider implementation to allow them to plug into the VCR. By so doing, they allow the delivery of personalized content through WebLogic Portal.

Portal users can also browse and manage the content in multiple repositories. This also lowers cost by allowing Portal developer to learn and use a single API versus understanding the complexities of the underlying repositories. However, in cases such as a home grown solution, customers may need to provide their own Content SPI implementation. This document provides the background necessary to accomplish this task.

Introduction

The BEA Virtual Content Repository (VCR) is designed to easily facilitate plugging in existing content repositories into BEA WebLogic

Portal. Adding your repository to the VCR gives you the ability to utilize the Portal interaction management federated content search and the content administration tools all on your existing content.

The goal of this document is to convey the necessary steps for a customer to implement the BEA Content Management SPI and plug the implementation into the VCR. This SPI should not be tied to any particular underlying architecture, data source or even protocol. This means it should be applicable to a wide range of implementations including database-centric, file-system-based and network protocol-based repositories.

The SPI implementation will run inside a WebLogic Portal Server enterprise application and may be clustered and also may be deployed multiple times with different configuration parameters. It is up to the SPI implementer to understand how their implementation will affect scalability and performance for WebLogic Portal. This white paper will point out particular areas of concern where different implementations strategies may adversely affect overall scalability.

SPI Structure

The SPI domain is modeled around a set of value objects. These value classes are created, read, updated and deleted through a set of service interfaces. The service interfaces are accessed through a set of repository connection interfaces that are the gateway to the SPI implementation.

The model tries to be as open as possible for the limited amount of functionality and relies on the SPI implementation to throw appropriate Exceptions if the attempted operations are not allowed in the repository. For example, the SPI allows for "content" to be added to "folders", if the repository does not allow that behavior then it should throw a RepositoryException stating so.

If a method is not implemented at all, then a UnsupportedRepositoryOperationException should be thrown. Implementing the BEA Content SPI involves implementing the repository connection and service interfaces that reside in com.bea.content.spi.

Value Classes

The content value objects are composed of Nodes that contain Properties and ObjectClasses that define the shape of the Nodes. Value objects are in-memory objects to the client of the SPI and thus can only be updated through the service interfaces. A default implemenetation for the value classes is provided in com.bea.content. They may be extended if necessary, but it is not advised.

ContentEntity

ConentEntity is the superclass for most of the value classes. It defines that any subclass will be Serializable and will also contain a unique id.

ID

ID uniquely identifies a ContentEntity in the system. It consists of both a repositoryName (used by management layer) and also a uid (used by repository layer). The management layer ensures that both the ID is never null and that its uid is never null when referring to a ContentEntity that exists in the system.

Node

This is a container for other Nodes and also Properties. A Node can be of type Hierarchy or Content. Hierarchy Nodes can contain both Hierarchy and Content Nodes, while Content Nodes can only contain other Content Nodes.

Nodes have a few system properties defined directly on the Node class including createDate, createdBy, modifiedDate and ModifiedBy. Nodes also have user or application defined Property objects. In order for a Node to contain Property Objects, it must have have an ObjectClass (defined below) associated with it.

Property

Property is a name value pair that may contain zero or more Value objects. Its associated PropertyDefinition defines the shape of a Property. The association is an implicit relationship based on name.

Value

This is a generic wrapper for the real value, which may be a BinaryValue, Boolean, Calendar, Float, Long or String.

BinaryValue

BestValue contains a byte array that is the actual binary and also has the binary name, mime type and size. A BinaryValue may not always contain the actual byes for performance reasons. In order for a consumer of the SPI to get the actual bytes, they should use the NodeOps service interface.

ObjectClass

ObjectClass defines the schema for a Node and contains a name that uniquely identifies it within a repository and also an array of PropertyDefinitions. ObjectClasses are stand-alone and do not provide an inheritance or containment structure.

As such, if the repository implementing the SPI has an inheritance or containment model, it must transform the requested ObjectClass into a single ObjectClass with all PropertyDefinitions found in all super ObjectClasses. If Nodes are associated to an ObjectClass care should be given to updating as there may be undesired outcomes.

Each ObjectClass may contain a primary PropertyDefintion. This allows the model to distinguish between content and meta-content. The primary PropertyDefinition indicates which Property on a Node represents the true content.

PropertyDefinition

PropertyDefinition defines the shape of a Property including its name, type, whether it is single-valued or multi-valued, read only, and mandatory. It also may define a set of PropertyChoices and whether the Property value is restricted to these choices. The validity of a PropertyDefinition is left to the SPI implementation with a few exceptions. The management layer requires the following rules be in place, even though they are not apparent in the class structure. These rules are enforced during the creation of a PropertyDefintion by the management layer, and the value model is not considered valid if the conditions are not true for PropertyDefintions (and the corresponding Properties) that are retrieved from an SPI implementation

- If the PropertyDefinition contains a reference, it may not be multi-valued, or binary.
- If the PropertyDefinition is Binary, it may not be multi-valued or restricted and may only have one PropertyChoice.
- If the PropertyDefinition is Boolean, it may not be multi-valued.

PropertyChoice

This defines a value choice for a Property and also whether it is a default for the Property. A PropertyChoice may be of any type that is supported by a Property.

Service Interfaces

Service Interfaces are used to perform CRUD on value interfaces such as creating a node or updating a property to a node. Service interfaces, in general, operate on the entity's id, such as Node id. This id can be a uuid, database id or the elements path as long as it uniquely identifies it to the repository.

Service interfaces are expected to be transactional (where necessary), thread safe and scalable. There may be one implementation for all interfaces, or a separate implementation for each service interface. All service methods throw a general com.bea.content. RepositoryException. The SPI also includes a few standard exceptions that inherit from RepositoryException.

NodeOps

NodeOps is the service interface for performing operations on Nodes and their Properties. This includes creating, updating, deleting, copying, moving and renaming.

ObjectClassOps

ObjectClassOps is the service interface for performing operations on ObjectClasses and their PropertyDefinitions and PropertyChoices. This includes creating, updating and retrieving.

SearchOps

SearchOpsis the service interface for performing searches for Nodes. The search is based on the Search object defined in com.bea.content. expression along with the expression structure in the com.bea.p13n.expression package.

A search is for any Node that matches the Property and/or ObjectClass definitions in the Expression. There are system properties that may be part of the search and are defined in the Search class. http://e-docs.bea.com/wlp/docs70/dev/exppkg.htm#1003588

Repository Connection Interfaces

Repository Connection Interfaces include Repository, Ticket and PasswordCredential. In order for a client of the SPI to connect to the services, the repository connection interfaces must be implemented. These interfaces allow an SPI implementation to be plugged into the BEA content management framework.

The repository connection interfaces are responsible for accepting a PasswordCredential object, which includes a username and granting the client access to the various service interfaces.

Repository

Repository defines the configuration for a content repository and also provides access to the content repository services through the Ticket. It must be implemented as a J2SE class such that "new" may be called on it.

The setProperties method on the Repository will be called on initialization so the Repository instance will always have its configuration properties before it is asked for a ticket. The Repository implementation may be configurable to allow for multiple instances to be deployed against different repositories.

Ticket

Ticket is the interface given back to the client after calling connect on a Repository with Credentials that are authenticated. It is the gateway to the content repository services.

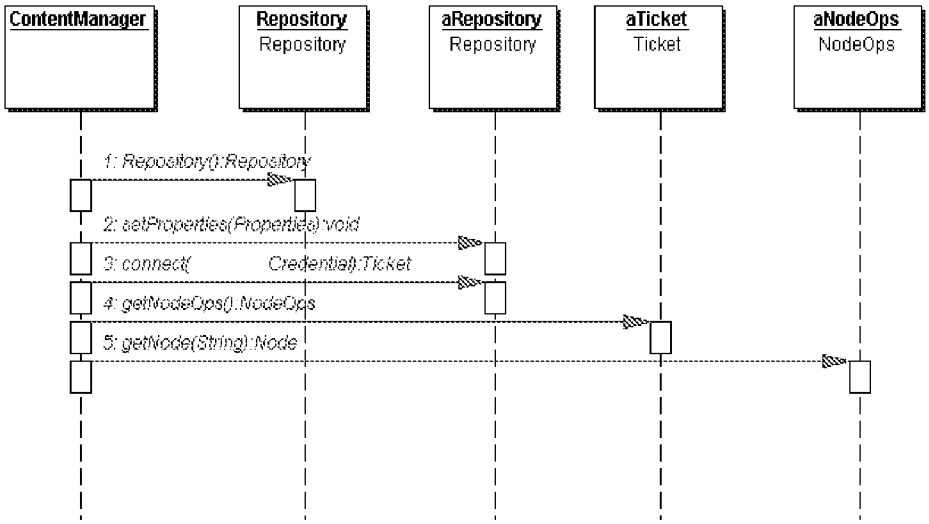
Credentials

Credentials defines the credentials for a user attempting access to the content repository. Both authentication and authorization may be based on the Credentials. It is necessary for the SPI implementation to authenticate and authorize based on username, because password is not present in the PasswordCredential. If the username is null then the authentication should be based on an anonymous user.

Connection Sequence

It is important to note that the ContentManager will call the SPI on a need to know basis. The SPI implementation can either take advantage of this by lazy loading, or it may load up front if that is more advantageous to the SPI or the application using it.

Figure 1 Connection sequence



For example, both the `getNodes` and `getNode` call may either return "light" versions of the Node that don't include the Properties, or they may return "full" versions of the Node that do include the Properties. If the Properties aren't part of the Node, when the ContentManager needs the Properties it will go back to the SPI to retrieve them. The InputStream for Binary Properties should only be returned on the getBytes () call, otherwise it won't get closed properly.

An example of a Repository implementation is that is called by the Virtual ContentManager is:

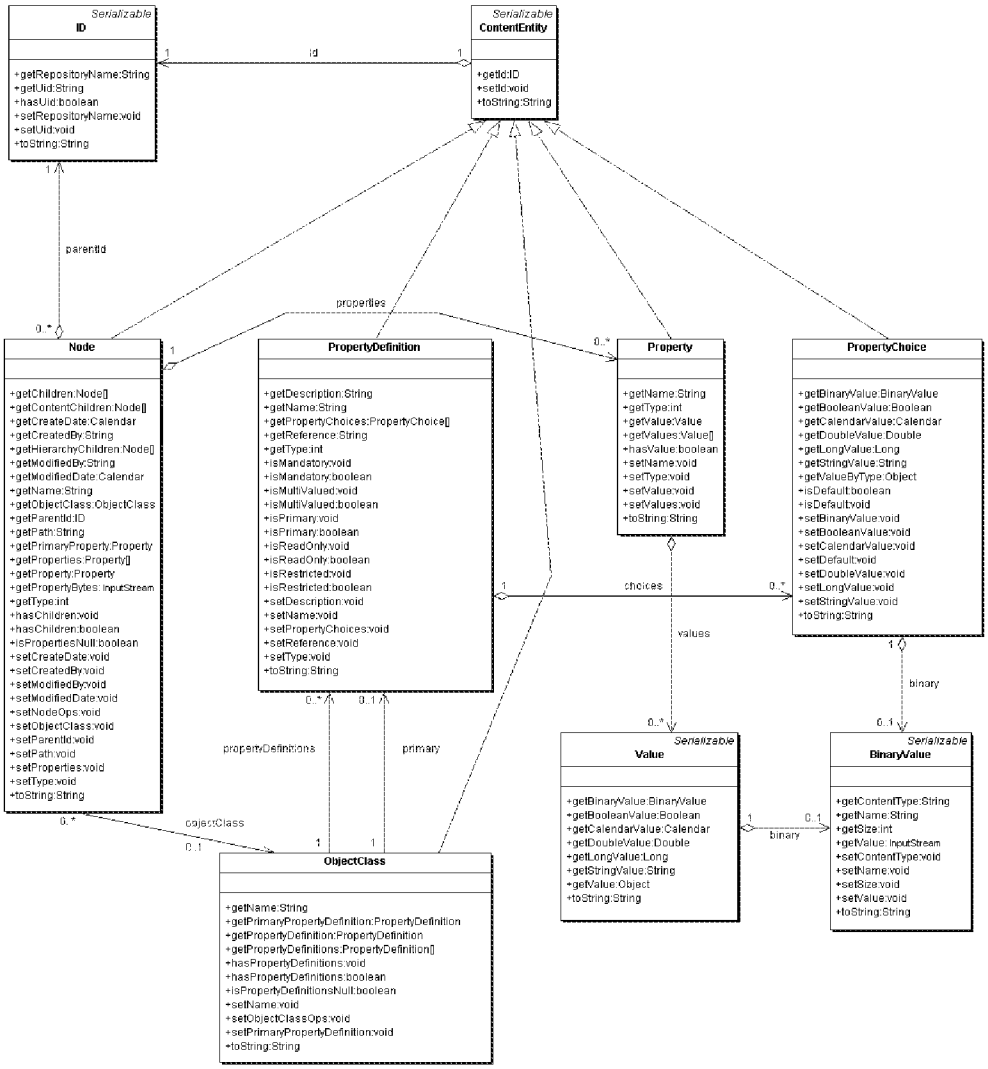
```
public class RepositoryImpl implements Repository
{
  Properties properties;
  String name;

  public Ticket connect(Credentials credentials)
  {
    return new TicketImpl(credentials, properties);
  }
  public Ticket connect(String userName, String password)
  {
    Subject subject = com.bea.p13n.security.Authentication.
    getCurrentSubject();
    Credentials credentials = new Credentials(subject);
    return new TicketImpl(credentials, properties);
  }
  public Properties getProperties()
  {
    return properties;
  }
  public void setProperties(Properties properties)
  {
    this.properties = properties;
  }
  public String getName()
  {
    return name;
```

```
    }
    public void setName(String name)
    {
      this.name = name;
    }
}
```

Value Model

Figure 2 shows the value model.

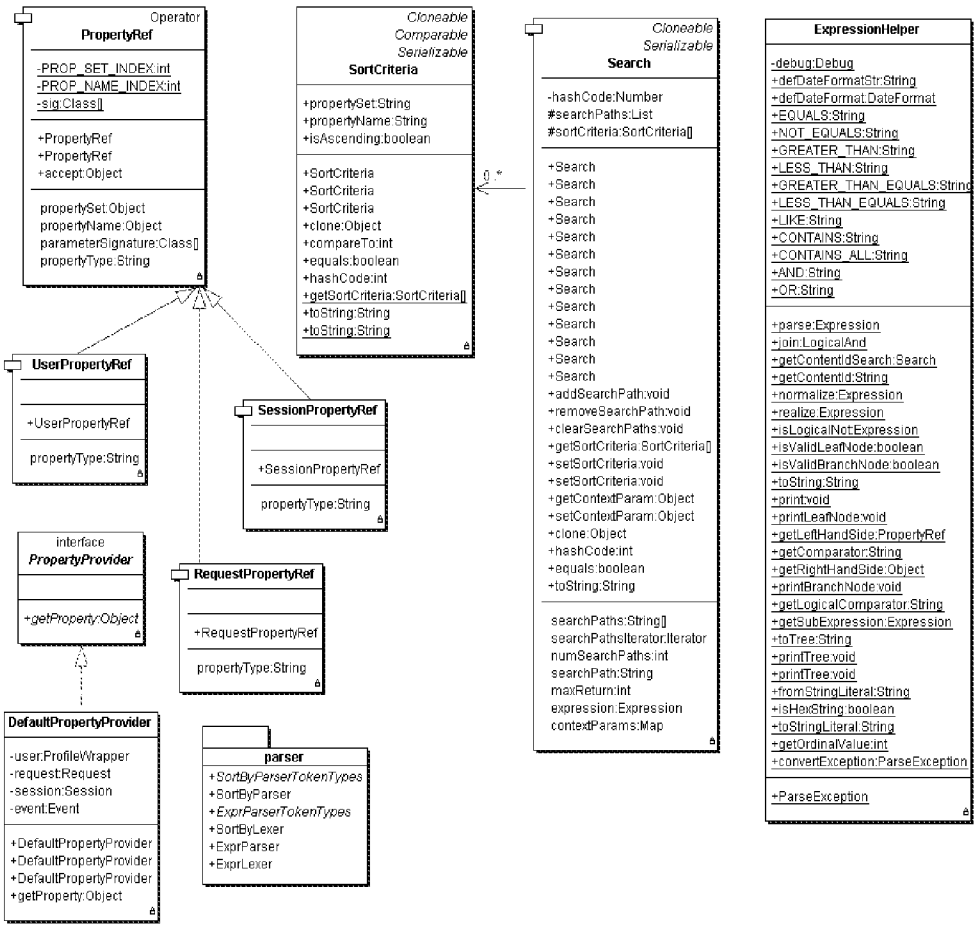
Figure 2 Value model



Configuring the Expression Model

Figure 3 shows the expression model.

Figure 3 Expression Model



Configuring the Repository

Once you have implemented the SPI you'll need to configure the Repository through the Content Administration Tools. Generally, your existing repository will require special configuration parameters in order for the VCR to connect to it. These properties may be added as key/value pairs through the Administration Tools, much like the BEA Repository utilizes. The properties that are added through the content administration tools are passed to your SPI implementation when the VCR connects to your repository as described above in the Connection Sequence. After configuring the repository, you can start utilizing it in BEA WebLogic Portal. For more information regarding the SPI, please review the Content SPI JavaDoc.

back to top previous next